08 | MQTT：在实践中掌握一个通信协议

2020-11-25 郭朝斌

物联网开发实战 进入课程 >

讲述：郭朝斌
时长 20:14 大小 18.53M

你好，我是郭朝斌。

在基础篇的第 3 讲中，我介绍了几种物联网系统中常用的网络协议。它们是物联网设备之间沟通的“语言”，使用同一种语言，双方才能通信成功。

那么，在这么多网络协议当中，最流行的是哪一种呢？毫无疑问，是 MQTT 协议，它甚至已经成为物联网系统事实上的网络协议标准。如果你想从事物联网开发，就一定要掌握 MQTT。

所以这一讲，我就会带你了解 MQTT，在实践中熟悉它的特性。

体验 MQTT

为了让你对 MQTT 有一个直观的印象，我先带你体验一下它的通信过程，

第一步是安装 hbmqtt，它是一个开源的基于 Python 语言的 MQTT Broker 软件，正好包括我们需要使用一些工具。跟其他选择相比，这个软件安装起来非常方便，因为它在 Python 的 PYPI 软件仓库中就有，所以你通过 pip 命令就可以安装。这也是选择使用它的主要原因。

不过要注意的是，hbmqtt 是基于 Python3 实现的，因此这里使用的是 pip3 工具。

```
pip3 install hbmqtt
```

安装完成后，我们就可以使用 hbmqtt 中提供的 hbmqtt_sub 和 hbmqtt_pub 这两个命令行工具了。通过名字，你应该也可以看出 hbmqtt_sub 可以充当**订阅者**的角色；hbmqtt_pub 可以作为消息的**发布者**。

至于订阅者和发布者之间的经纪人，也就是 MQTT Broker，我们使用 Eclipse 免费开放的在线 Broker 服务。打开链接，你可以看到关于端口的介绍信息，加密和非加密方式都支持，而且还有基于 Websocket 的实现，这对基于前端网页的应用来说是非常有利的。

我们先使用 1883 端口的非加密方式，然后为消息传输确定一个主题（Topic）。主题确定了消息的类别，用于消息过滤。比如我们待会儿要测试的消息，属于极客时间平台的物联网课程，所以主题可以设为“/geektime/iot”。

```
/geektime/iot
```

接着，我们在电脑的终端界面输入下面的命令，就可以订阅这个主题消息：

```
hbmqtt_sub --url mqtt://mqtt.eclipse.org:1883 -t /geektime/iot
```

如果你想了解一些命令的执行细节，可以在上面的命令中加上 “-d” 参数。

现在，我们启动另外一个终端界面，通过 hbmqtt_pub 发布一个 “/geektime/iot” 主题的消息：

```
1 hbmqtt_pub --url mqtt://mqtt.eclipse.org:1883 -t /geektime/iot -m Hello,World!
```

复制代码

通过 Eclipse 的开放 Broker 作为“经纪人”，消息被传输到了我们通过 hbmqtt_sub 运行的订阅者那里。下图是我的终端界面上运行的结果，一个完整的消息传输过程就这样完成了。

```
→  /Users/rubin/study/iot/geektime  >hbmqtt_sub --url mqtt://mqtt.eclipse.org:1883 -t /geektime
/iot
[2020-11-01 10:15:58,599] :: INFO - Finished processing state new exit callbacks.
[2020-11-01 10:15:58,599] :: INFO - Finished processing state connected enter callbacks.
Hello,World!
```

MQTT 的生态很完善

初步体验之后，不知道你是不是有这样的感觉：原来使用 MQTT 也没有那么难啊？

确实不难，甚至可以说很简单，这主要得益于 MQTT 协议出现的时间很久远。当然，时间久远本身不是优势，能力强才是。这些长期的使用和积淀使得 MQTT 的生态非常完善，而生态是技术标准能够主导行业的关键。所以你在使用 MQTT 的时候会觉得很方便，可供挑选的方案有很多。

比如在上面的体验中，你要安装 MQTT，可以直接找到 hbmqtt 这样的项目拿来用。它的背后有 Eclipse 基金会的支持。

类似的 MQTT Broker 软件，你还可以选择基于 C 语言的 Mosquitto，基于 Erlang 语言的 VerneMQ 等。

至于 MQTT 的客户端（Client）实现，也有成熟的 Python、C、Java 和 JavaScript 等各种编程语言的开源实现，供你参考、使用，比如 Eclipse Paho 项目。

而且，还有很多商业公司在持续运营功能更丰富、支持更完备的商业版 Broker 实现，比如提供高并发能力的集群特性、方便拓展的插件机制等。这些会大大提高我们技术开发者的

工作效率。比如中国一个团队开发、维护的 EMQ X，它已经完整地支持 MQTT5.0 协议。

另外，生态完善还有一个好处，那就是作为开发者，当你遇到难题时，可以很方便地找到很多相关的资料；就算资料解决不了问题，你还可以去社区中提问，寻求高手的帮助。这在实际工作中非常有用。顺便提一下，除了老牌的 Stack Overflow，你还可以关注一下 GitHub 的 Issues 模块，因为在那里可以找到很多专家。

MQTT 自身的“基因”很强大

当然，阿里云、华为云、腾讯云和微软 Azure 这些大厂，之所以不约而同地选择 MQTT 协议作为物联网设备的“第一语言”，不仅是因为 MQTT 的生态完善，MQTT 协议本身的优秀设计也是重要的因素。

它在设计上的优点体现在哪呢？我想主要有五个方面：

1. 契合物联网大部分应用场景的**发布 - 订阅模式**。
2. 能够满足物联网中资源受限设备需要的**轻量级特性**。
3. 时刻关注物联网设备**低功耗需求的优化设计**。
4. 针对物联网中多变的网络环境提供的**多种服务质量等级**。
5. 支持在物联网应用中越来越被重视的**数据安全**。

接下来，我分别为你剖析一下。

发布 - 订阅模式

刚才我们体验的通信过程，是一个发布者和一个订阅者的情况。在这之后，你可以再打开一个终端界面，重复和之前一样的命令，再启动一个订阅者。

```
1 hbmqtt_sub --url mqtt://mqtt.eclipse.org:1883 -t /geektime/iot
```

复制代码

现在，这两个订阅者都订阅了“/geektime/iot” 主题的消息。

然后，你再次使用 hbmqtt_pub 发送消息，就可以看到两个订阅者都收到了同样的消息，这是**发布 - 订阅模式**的典型特征。

因为采用了发布 - 订阅模式，MQTT 协议具有很多优点，比如能让一个传感器数据触发一系列动作；网络不稳定造成的临时离线不会影响工作；方便根据需求动态调整系统规模等。这使得它能满足绝大部分物联网场景的需求。

轻量级协议：减少传输数据量

MQTT 是一个**轻量级**的网络协议，这一点也是它在物联网系统中流行的重要原因。毕竟物联网中大量的都是计算资源有限、网络带宽低的设备。

这种“轻量级”体现在两个方面。一方面，MQTT 消息采用**二进制的编码格式**，而不是 HTTP 协议那样的文本的表述方式。

这有什么好处呢？那就是可以充分利用字节位，协议头可以很紧凑，从而尽量**减少需要通过网络传输的数据量**。

比如，我们分析 HTTP 的一个请求抓包，它的消息内容是下面这样的（注意：空格和回车、换行符都是消息的组成部分）：

复制代码

```
GET /account HTTP/1.1    <--注释：HTTP请求行
Host: time.geekbang.com  <--注释：以下为HTTP请求头部
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:81.0) Gecko/20100
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
                         <--注释：这个空行是必须的，即使下面的请求体是空的
```

在 HTTP 协议传输的这段文本中，每个字符都要占用 1 个字节。而如果使用 MQTT 协议，一个字节就可以表示很多内容。下面的图片展示了 MQTT 的固定头的格式，这个固定头只有 2 个字节：

Bit	7	6	5	4	3	2	1	0
字节1	MQTT数据类型				数据包标识位			
字节2	延续位	可变头部和消息体的字节个数						

你可以看到，第一个字节分成了高 4 位（4～7）和低 4 位（0～3）；低 4 位是数据包标识位，其中的每一比特位又可以表示不同的含义；高 4 位是不同数据包类型的标识位。

第二个字节表示数据包头部和消息体的字节共个数，其中最高位表示有没有第三字节存在，来和第二个字节一起表示字节共个数。

如果有第三个字节，那它的最高位表示是否有第四个字节，来和第二个字节、第三个字节一起表示字节总个数。依此类推，还可能有第四个字节、第五个字节，不过这个表示可变头部和消息体的字节个数的部分，最多也只能到第五个字节，所以可以表示的最大数据包长度有 256MB。

比如，一个请求建立连接的 CONNECT 类型数据包，头部需要 14 个字节；发布消息的 PUBLISH 类型数据包头部只有 2～4 个字节。

轻量级的另一方面，体现在消息的具体**交互流程设计非常简单**，所以 MQTT 的交互消息类型也非常少。为了方便后面的讲解，我在这里整理了一个表格，总结了 MQTT 不同的数据包类型的功能和发消息的流向。

从表格可以看出，MQTT 3.1.1 版本一共定义了 14 种数据包的类型，在第一个字节的高 4 位中分别对应从 1 到 14 的数值。

信息类型	数值(二进制)	流方向	功能
CONNECT	0001	Client ---> Broker	Client 请求连接Broker
CONNACK	0010	Client <--- Broker	Broker确认连接请求
PUBLISH	0011	Client <--> Broker	发布主题消息
PUBACK	0100	Client <--> Broker	主题消息的发送方收到的确认消息
PUBREC	0101	Client <--> Broker	QoS2 中第一步发布确认
PUBREL	0110	Client <--> Broker	QoS2 中第二步释放确认
PUBCOMP	0111	Client <--> Broker	QoS2 中第三步完成确认
SUBSCRIBE	1000	Client ---> Broker	Client 请求订阅主题消息
SUBACK	1001	Client <--- Broker	Broker 确认订阅请求
UNSUBSCRIBE	1010	Client ---> Broker	Client 请求取消主题消息订阅
UNSUBACK	1011	Client <--- Broker	Broker 确认取消消息订阅
PINGREQ	1100	Client ---> Broker	Client 的PING 请求
PINGRESP	1101	Client <--- Broker	Broker 的PING 回复
DISCONNECT	1110	Client ---> Broker	Client 中断连接请求

低功耗优化：节约电量和网络资源

除了让协议足够轻量，MQTT 协议还很注重**低功耗**的优化设计，这主要体现在对能耗和通信次数的优化。

比如，MQTT 协议有一个 **Keepalive 机制**。它的作用是，在 Client 和 Broker 的连接中断时，让双方能及时发现，并重新建立 MQTT 连接，保证主题消息的可靠传输。

这个机制工作的原理是：Client 和 Broker 都基于 Keepalive 确定的时间长度，来判断一段时间内是否有消息在双方之间传输。这个 Keepalive 时间长度是在 Client 建立连接时设置的，如果超出这个时间长度，双方没有收到新的数据包，那么就判定连接断开。

虽然 Keepalive 要求一段时间内必须有数据包传输，但实际情况是，Client 和 Broker 不可能时时刻刻都在传输主题消息，这要怎么办呢？

MQTT 协议的解决方案是，定义了 PINGREQ 和 PINGRESP 这两种消息类型。它们都没有可变头部和消息体，也就是说都只有 2 个字节大小。Client 和 Broker 通过分别发送 PINGREQ 和 PINGRESP 消息，就能够满足 Keepalive 机制的要求。

我猜你也想到了，如果要一直这样“傻傻地”定期发送消息，那也太浪费电量和网络资源了。所以，如果在 Keepalive 时间长度内，Client 和 Broker 之间有数据传输，那么

Keepalive 机制也会将其计算在内，这样就不需要再通过发送 PINGREQ 和 PINGRESP 消息来判断了。

除了 Keepalive 机制，MQTT 5.0 中的**重复主题特性**也能帮助我们节省网络资源。

Client 在重复发送一个主题的消息时，可以从第二次开始，将主题名长度设置为 0，这样 Broker 会自动按照上次的主题来处理消息。这种情况对传感器设备来说十分常见，所以这个特性在工作中很有实际意义。

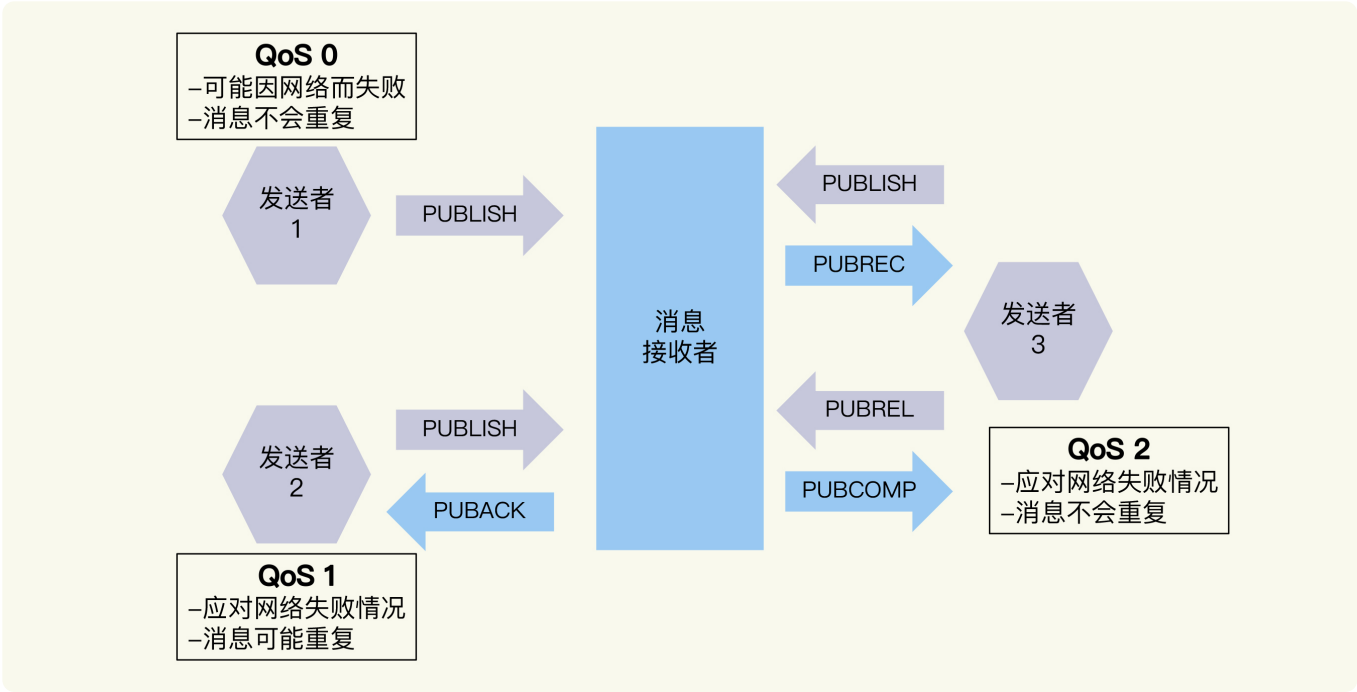
3 种 QoS 级别：可靠通信

除了计算资源有限、网络带宽低，物联网设备还经常遇到网络环境不稳定的问题，尤其是在移动通信、卫星通信这样的场景下。比如共享单车，如果用户已经锁车的这个消息，不能可靠地上传到服务器，那么计费就会出现错误，结果引起用户的抱怨。这样怎么应对呢？

这个问题产生的背景就是不稳定的通信条件，所以 MQTT 协议设计了 3 种不同的 QoS（Quality of Service，服务质量）级别。你可以根据场景灵活选择，在不同环境下保证通信是可靠的。

这 3 种级别分别是：

1. QoS 0，表示消息**最多**收到一次，即消息可能丢失，但是不会重复。
2. QoS 1，表示消息**至少**收到一次，即消息保证送达，但是可能重复。
3. QoS 2，表示消息**只会**收到一次，即消息有且只有一次。



我用一张图展示了它们各自的特点。可以看到，QoS 0 和 QoS 1 的流程相对比较简单；而 QoS 2 为了保证有且只有一次的可靠传输，流程相对复杂些。

正常情况下，QoS 2 有 PUBLISH、PUBREC、PUBREL 和 PUBCOMP 4 次交互。

至于“不正常的情况”，发送方就需要重复发送消息。比如一段时间内没有收到 PUBREC 消息，就需要再次发送 PUBLISH 消息。不过要注意，这时要把消息中的 “重复”标识设置为 1，以便接收方能正确处理。同样地，如果没有收到 PUBCOMP 消息，发送方就需要再次发送 PUBREL 消息。

剖析到这里，MQTT 协议本身的主要特性我就介绍完了，我们已经为在实战篇编写 MQTT 的相关通信代码做好了准备。但是，我还想跟你补充一个跟生产环境有关的知识点，那就是**数据安全传输**。

安全传输

说到安全传输，首先我们需要验证 Client 是否有权限接入 MQTT Broker。为了控制 Client 的接入，MQTT 提供了**用户名 / 密码**的机制。在建立连接过程中，它可以通过判断用户名和密码的正确性，来筛选有效连接请求。

但是光靠这个机制，还不能保证网络通信过程中的数据安全。因为在明文传输的方式下，不止设备数据，甚至用户名和密码都可能被其他人从网络上截获而导致泄漏，于是其他人

就可以伪装成合法的设备发送数据。所以，我们还需要通信加密技术的支持。

MQTT 协议支持 **SSL/TLS** 加密通信方式。采用 SSL/TLS 加密之后，MQTT 将转换为 MQTTS。这有点类似于 HTTP 和 HTTPS 的关系。

我们只要将前面测试的命令修改一下，将 “mqtt://” 改为 “mqtts://”，端口改为 8883，就可以用 SSL/TLS 加密通信方式连接到 Eclipse 提供的开放 Broker。但是我最近发现，它的 SSL 证书已经过期了，因此连接会失败。（你在学习这一讲的时候，可以再试试，万一又更新了呢？）

所以，我再提供另一个方式，供你测试使用。

我们输入“mqtts://test.mosquitto.org:8883”，把开放 Broker 切换到这个链接，从链接中下载一个客户端证书，然后通过下面的命令订阅主题消息：

```
1 hbmqtt_sub --url mqtts://test.mosquitto.org:8883 -t /geektime/iot --ca-file ~/
```

复制代码

接着，我们再通过下面的命令测试发布消息：

```
1 hbmqtt_pub --url mqtts://test.mosquitto.org:8883 -t /geektime/iot -m Hello,Wor
```

复制代码

最后在运行 hbmqtt_sub 命令的终端，就可以看到 Hello,World! 的消息：

```
➜  /Users/rubin/study/iot/geektime  >hbmqtt_sub --url mqtts://test.mosquitto.org:8883
-t /geektime/iot --ca-file ~/Downloads/mosquitto.org.crt
[2020-11-23 16:20:54,015] :: INFO - Finished processing state new exit callbacks.
[2020-11-23 16:20:54,015] :: INFO - Finished processing state connected enter callback
s.
Hello,World!
```

小结

总结一下，在这一讲中，我带你体验了使用 MQTT 协议的通信过程，同时也为你介绍了 MQTT 协议的几个特点。

1. MQTT 协议的生态很好。比如，MQTT 协议的代码实现非常丰富，C 语言的有 Mosquitto，Python 语言有 Eclipse hbmqtt，而且有商业公司在运营相关软件解决方案。这表明 MQTT 协议很成熟。
2. MQTT 协议采用了适合物联网应用场景的发布 - 订阅模式。当然，我也提到了 MQTT 5.0 中同样增加了请求 - 响应模式，便于部分场景中的开发使用。
3. MQTT 协议采用二进制的消息内容编码格式，协议很精简，协议交互也简单，这些特点保证了网络传输流量很小。所以客户端（包括发布者和订阅者角色）的代码实现可以短小精悍，比如 C 语言的实现大概只占 30KB 的存储空间，Java 语言也只需要 100KB 左右大小的代码体积。
4. MQTT 协议在设计上考虑了很多物联网设备的低功耗需求，比如 Keepalive 机制中精简的 PINGREQ 和 PINGRESP 这两种消息类型，还有 MQTT 5.0 中新增加的重复主题特性。这也再次印证了 MQTT 的定位非常明确，那就是专注于物联网场景。
5. MQTT 在消息的可靠传输和安全性上，也有完整的支持，可以说“简约而不简单”。

因此，在你为物联网系统选择网络协议时，MQTT 可以作为重点考察对象。在实战篇的动手实验中，我们将使用 MQTT 协议传输设备数据，相信你已经做好了准备。

思考题

最后，我想给你留一个实践作业和一道思考题。

在介绍 QoS 等级的时候，我没有结合网络抓包来说明消息的交互过程。所以，我想请你在课后使用 Wireshark 工具，实际分析一下各个 QoS 级别的数据包类型。

同时，也请你思考一下，如果 Client 发布消息时选择的 QoS 等级是 1 ，而订阅者在订阅这个主题消息时选择的 QoS 等级是 2 ，这种情况下 Broker 会怎么处理呢?

欢迎你在留言区写一写你的实践和思考的结果，也欢迎你将这一讲分享给你的朋友一起交流学习。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 07 | 零配置组网：设备如何发现彼此?

下一篇 09 | 边缘中心：物联网网关有多重要?

精选留言 (5)

写留言

InfoQ_Albert
2020-11-26

经实践SSL 证书已经可用，利用mqtts发布和订阅消息成功。

作者回复: great！感谢你的留言

1

Felix
2020-12-02

发布着选择忽略PUBACK?
如果反过来，发布者选择QoS等级是2，订阅者选择QoS等级是1呢？发布者就会重复发布同样的消息?

———————————

如果 Client 发布消息时选择的 QoS 等级是 1 ，而订阅者在订阅这个主题消息时选择的 ...
展开

作者回复: 实际抓包试一下，不复杂，掌握的更深

YueShi
2020-11-27

Client 在重复发送一个主题的消息时，可以从第二次开始，将主题名长度设置为 0，这样

Broker 会自动按照上次的主题来处理消息。这种情况对传感器设备来说十分常见，所以这个特性在工作中很有实际意义。

这段没有太懂, 有老哥给提示一下吗? 多谢了

展开

作者回复: 比如发生是主题是“/geektime/iot”，第一次需要在信息体中包含这个主题的字符，第二次就可以省掉这个主题名字了。

il李li

2020-11-25

MQTT Broker 根据客户端建立连接时选择的QoS级别与客户端交互，也即是与发布消息的Client按照QoS=1 （至少一次）处理，broker将消息推送给订阅这个主题的Clien时选择 QoS=2（只有一次）来处理

作者回复: broker只是中间人，如何保证订阅者可以QoS2呢?

ACK

2020-11-25

作为智能锁嵌入式开发者，觉得老师讲的东西很受用😃

作者回复: 谢谢你的留言：）

1